ROBERT A. VAN NEST (SBN 84065)
rvannest@kvn.com
CHRISTA M. ANDERSON (SBN 184325)
canderson@kvn.com
KEKER & VAN NEST LLP
633 Battery Street
San Francisco, CA 94111-1809
Telephone: (415) 391-5400
Facsimile: (415) 397-7188

SCOTT T. WEINGAERTNER (*Pro Hac Vice*)
sweingaertner@kslaw.com
ROBERT F. PERRY
rperry@kslaw.com
BRUCE W. BABER (*Pro Hac Vice*)
bbaber@kslaw.com
KING & SPALDING LLP
1185 Avenue of the Americas
New York, NY 10036-4003
Telephone: (212) 556-2100
Facsimile: (212) 556-2222

DONALD F. ZIMMER, JR. (SBN 112279)
fzimmer@kslaw.com
CHERYL A. SABNIS (SBN 224323)
csabnis@kslaw.com
KING & SPALDING LLP
101 Second Street – Suite 2300
San Francisco, CA 94105
Telephone: (415) 318-1200
Facsimile: (415) 318-1300

IAN C. BALLON (SBN 141819)
ballon@gtlaw.com
HEATHER MEEKER (SBN 172148)
meekerh@gtlaw.com
GREENBERG TRAURIG, LLP
1900 University Avenue
East Palo Alto, CA 94303
Telephone: (650) 328-8500
Facsimile: (650) 328-8508

Attorneys for Defendant
GOOGLE INC.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.

Plaintiff,

v.

GOOGLE INC.

Defendant.

Case No. 3:10-cv-03561-WHA

Honorable Judge William Alsup

GOOGLE'S REPLY TO ORACLE'S PROPOSED CLAIM CONTRUCTIONS

Pursuant to the December 21, 2011 *Interim Pretrial Conference* (Dkt. No. 654 at 154:12–155:20), Google hereby submits this response to Oracle's supplemental claim construction brief filed December 9, 2011 (Dkt. No. 645) ("Oracle Br."). Per the parties' agreement (Dkt. No. 654 at 154:20–155:20), Google relies on its prior briefing with respect to the "computer-readable medium" term of claim 14 of the '476 patent. (*See* Dkt. Nos. 96–97, 102–03.)

A. The '720 Patent: "obtain a representation . . . from a source definition . . ."

Oracle's arguments rely on two fundamentally flawed premises: (i) that Google's proposed definition reads out the preferred embodiment, and (ii) that a cherry-picked, non-technical dictionary definition overrides the patent specification. Neither is true. In fact, Oracle's proposed construction at best renders the claim language superfluous, and at worst is in conflict with the intrinsic evidence from the specification.

1. Google's—and not Oracle's—proposed construction captures the preferred embodiment by including a class loader capable of compiling source code.

Oracle incorrectly claims that Google and its expert "rely on a misreading of a sentence fragment in the '720 specification" to arrive at a construction that "excludes the preferred embodiments." (Oracle Br. at 5:21–22, 5:13–14.) But Google's proposed construction does not exclude systems that preload Java class files produced by a Java compiler; it explicitly includes them. Where Google's proposed construction differs, however, is that it gives meaning to the requirement that "at least one" of those classes be "from a source definition provided as object oriented program code." In other words, source code. As such, Google's proposed construction ***includes*** the preferred embodiments disclosed in column 9 of the '720 patent specification.

Oracle concedes that the operation of the preferred preloader is described in column 9 of the '720 patent under the heading "Routine for Preloading Classes," which identifies three mechanisms for preloading classes. (*See* Dkt. No. 647, Google Br. at 4:16–5:7.) The preloader first attempts to locate "the class" in the system class dictionary. ('720 patent at 9:42–43.) If not found in the system class dictionary, the preloader then looks for "the class" in the file system. (*Id.* at 9:45–47.) If not found in the file system, then the preloader "attempts to load the bytes for the class from the source associated with the [core class loaders]" and creates "an instance of the

class . . . by compiling the source . . .”[1] (*Id.* at 9:48–54.) It is this source code compilation process[2] that is captured by the express claim language that refers to “obtaining a representation of at least one class from a source definition provided as object oriented program code.”

Oracle mistakenly contends that the phrase “compiling the source” refers to compiling Java bytecode (which is not source code) with a just-in-time (JIT) compiler. (Oracle Br. at 6–7.) This argument is made of whole cloth: the ‘720 patent contains neither discussion of nor reference to JIT compilation. This argument fails for at least two technical reasons. First, because classes are preloaded into the master process before they are actually required by a user application in a child process (‘720 patent at 2:66–3:1, 10:5–14), compilation by the preloader is speculative and well in advance of execution. Put another way, because the ***preloader*** performs the step of “compiling the source,” the patent addresses an approach wholly different from a JIT compiler that “compiles an entire Java function ***just before it is called***,” i.e., during execution. (Oracle Br. at 6:24 (emphasis added).) Second, the ‘720 patent only uses the term “compiling” in conjunction with one of three loading mechanisms enumerated in the ‘720 patent. (*See, e.g.*, ‘720 patent at 9:48–54.) Oracle’s construction does not address (but rather obfuscates) this aspect of the claimed system. In contrast, Google’s proposed construction provides a clear explanation: two mechanisms load classes from precompiled bytecode while the third loads classes from source code that must first be compiled.

2. Oracle’s proposed definition conflicts with the specification.

Oracle presents two extrinsic dictionary definitions of “source” in opposition to Google’s proposed construction, neither of which can be correct in the context of the patent. First, Oracle advances the second of several definitions for “source” provided by Webster’s dictionary: “a point of origin or procurement.” (Oracle Br. at 1:25–27.) This definition renders the preferred

---

[1] Oracle argues that “bytes” indicates that the contents of “the source” must be bytecode. (Oracle Br. at 6:9–15.) This argument is specious: “bytes” simply refers to a unit of storage, and both source and object code are represented as bytes in a file. (*See* Fenton Decl. Ex. A.)

[2] The straightforward idea of a source code compiling loader was explained in a contemporary Java tutorial published by IBM in 2001. (Fenton Decl. Ex. B at 7 (“If the class is not available, but the source is, call the Java compiler to generate the class file.”).)

embodiments meaningless—if we replace the word "source" with Oracle's proposed definition of that word, then the sentence in the specification discussing compilation becomes nonsensical: "If successful (block 158), an instance of the class is created by compiling the <<point of origin or procurement>> and the class instance is installed in the system class dictionary (block 160)." ('720 patent at 9:51–54.) The same is true of substituting the definition of "source" found in the Microsoft dictionary ("a disk, file, document, . . ."), also advanced by Oracle. In contrast, Google's definition of "source code" ***can*** be meaningfully substituted into the '720 patent specification for "source": "If successful (block 158), an instance of the class is created by compiling the <<human-readable program statements . . . >> . . ." Thus, "source definition provided as object oriented program code" must refer to object-oriented source code.

3. Oracle's proposed construction renders the phrase at issue redundant.

As noted above, Oracle argues that the word "source" simply means "a point of origin or procurement." (Oracle Br. at 2.) Because the claim limitation recites "obtaining a representation of at least one class from a ***source*** definition," it is difficult, if not impossible, to imagine any system that "obtains a representation of at least one class" from anywhere other than the "point of origin or procurement" of that class. As such, Oracle's proposal renders the claim language meaningless, and cannot be correct. *See Curtiss-Wright Flow Control Corp. v. Velan, Inc.* 438 F.3d 1374, 1379–80 (Fed. Cir. 2006). The claim-at-issue in *Curtiss-Wright* was drawn to a mechanical device, and the claim term "adjustable" was construed according to its ordinary meaning. *See id.* But in reviewing the district court's claim construction, the Federal Circuit noted that it was "difficult, if not impossible, to imagine any mechanical device that is not 'adjustable,' under the ordinary meaning of that term adopted by the district court." *Id.* While the Federal Circuit "commend[ed] the district court's reluctance to narrow the claims," it ultimately found the construction flawed because it "renders that ['adjustable'] limitation meaningless." *Id.* The same is true here: a preloader by its very nature finds classes at "a point of origin or procurements." Thus construing the phrase "from a source definition provided as object oriented program code" as Oracle suggests adds nothing. In contrast, Google's proposed definition gives meaning to the additional language in such a way that captures the preferred

embodiment—runtime compilation of source code when classes have not yet been created.

4. Oracle's claim differentiation argument relies on a specific definition of the term "class file" that is unsupported by the intrinsic or extrinsic record.

Oracle insists that when claim 5 requires a file system to "maintain the source definition as a class file," the claim requires that the "class file" be "a compiled binary file satisfying the Java Virtual Machine Specification." (Oracle Br. at 3:10–11.) But this "requirement" is not part of the claim, and Oracle provides no intrinsic or extrinsic evidence to support such a narrow interpretation of the phrase "class file." The '720 patent neither defines the term "class file" nor incorporates the Java Virtual Machine Specification by reference, and refers to Java technologies only as "[a]n example of a suitable managed code platform . . ." ('720 patent at 3:6–10.)

Oracle's argument regarding graphic icons does not alter this result as Oracle is simply incorrect about the meaning of those icons. The icon for classes 36, a box with a wavy bottom, merely represents a document, or a file. (*See* Fenton Decl. Ex. C at 123, 144; Ex. D (showing a flowcharting template with the identical icon in a 2003 drafting program).)

B. The '205 Patent: "runtime"

Oracle's "ordinary meaning" construction of the term "runtime" ignores both the plain language of claim 1 and the entirety of the specification, and relies on unsupported and inaccurate technical conclusions. Oracle's attempt to expand the scope of the claims of the '205 patent beyond anything contemplated or disclosed by the patentee should be rejected.

1. Oracle presents no evidence of an "ordinary meaning."

Oracle provides no evidence that its proposed "ordinary meaning" construction reflects the common meaning of the term "runtime." (Oracle Br. at 9.) Indeed, the meaning of that term depends on the context in which it is used. Oracle fails in its five and a half pages of briefing to explain what a person of ordinary skill would understand the term to mean in the context of the '205 patent. Oracle provides only its present-day, litigation-driven interpretation of the term divorced from the context provided in both the claim itself and the specification. *See Phillips v. AWH Corp.*, 415 F.3d 1303, 1313 (Fed. Cir. 2005) (ordinary artisan deemed to read the claim term not only in the context of the claim, but in the context of the entire patent).

2. The language of claim 1 does not support Oracle's construction.

On the one hand, Oracle's proposed construction disregards the preamble of claim 1, which explicitly refers to "increasing the speed of virtual machine *instructions* at runtime." (*See* '205 patent, claim 1; *see also* Google Br. at 7–8.) On the other hand, Google's proposal looks to the preamble as the only other intrinsic use of the word "runtime." As such, the preamble provides insight into what the patentee intended when limiting the "generating" step to occur "at runtime," namely, during execution of the virtual machine ***instructions***. *See Pitney Bowes, Inc. v. Hewlett-Packard Co.*, 182 F.3d 1298, 1306 (Fed. Cir. 1999) (where meaning of claim term "is only discernable from the claim preamble . . . , it is essential that the court . . . construe the preamble and the remainder of the claim . . . as one unified and internally consistent recitation"). Notably, Google does not rely on the preamble as a limitation itself, as the term runtime is an explicit limitation of the "generating" step. (Oracle's reliance on *Catalina Mktg.* is therefore misplaced. (*See* Oracle Br. at 10.))

In disregarding the preamble, Oracle's construction renders the "at runtime" phrase meaningless. Oracle argues that "all that is required [in the 'generating, at runtime' step] is the generation of a new virtual machine instruction 'that can be' executed instead of the first virtual machine instruction." (Oracle Br. at 10.) If Oracle's argument is correct, then the "at runtime" phrase in this limitation is meaningless. This cannot be the case. *Curtiss-Wright*, 438 F.3d at 1379 (court should not construe a term in such a way that renders it meaningless).

3. The specification does not support Oracle's construction.

In its various citations to the '205 patent (and to other patents-in-suit), Oracle fails to show a single instance where a new virtual machine instruction is generated at a time other than when the virtual machine is executing virtual machine instructions. In fact, the specification consistently discloses the generation of a new virtual machine instruction ***during the execution of instructions of a program***. In particular, the patent describes the "go_native" instruction as the new virtual machine instruction. (*E.g.*, '205 patent at 7:23–26, 9:30–34.) The go_native instruction is generated whenever the interpreter decides to substitute a sequence of bytecodes with optimized native machine instructions, which the '205 patent calls a "snippet." (*E.g.*, '205

patent at 6:26–30 ("virtual machine instructions . . . may be compiled into native machine instructions in the form of a 'snippet'"); 7:49–57 ("the interpreter decides when to substitute a sequence of bytecodes with a snippet" and generates a go_native bytecode that specifies the generated snippet).) The specification is explicit that this process occurs during execution of the virtual machine instructions of the then-executing program. (*See* '205 patent at 9:30–34 ("As snippets are generated ***during program execution***, the new virtual machine instruction is executed . . ." (emphasis added)); 7:49–51 ("***During execution of an interpreted program***, the interpreter decides when to substitute a sequence of bytecodes with a snippet . . . [and then generate] a go_native bytecode . . . specifying the snippet." (emphasis added)).)

In fact, nowhere does the patent disclose generating a new virtual machine instruction at a time other than during program execution. Thus, this case is similar to *Edwards Lifesciences LLC v. Cook Inc.*, 582 F.3d 1322 (Fed. Cir. 2009). The issue in *Edwards* was whether the term "graft" was limited to "intraluminal grafts." *Id.* at 1328–29. The court held that it was, reasoning, *inter alia*, that "the only devices described in the specification are intraluminal, supporting an interpretation of [the term 'graft'] that is consistent with that description." *Id.* at 1329. So too here. The generation of the new go_native virtual machine instruction is consistently described as occurring while the program being optimized is in execution. Thus, this Court should construe "runtime" to be consistent with that description.

Finally, Oracle misreads the description of Figure 13 and incorrectly concludes that generation of a new virtual machine instruction is occurring during the disclosed initialization. (*See* Oracle Br. at 12 (quoting '205 patent at 12:15–26 and concluding that this passage "describes the technique of generating a new virtual machine instruction . . . while the virtual machine is up-and-running, before executing any virtual machines.").) To begin with, Oracle fails to explain why Figure 13 allegedly supports its proposed construction, as it is unclear why the "initialization" described with respect to Figure 13 should be considered "execution" of the virtual machine. In any event, the '205 patent describes Figure 13 as a bytecode table that is preferably generated "once when the Java virtual machine is initialized." ('205 patent at 13:3–8.) But the patent goes on to say that "[t]he bytecode table may include a pointer to snippet code

1061 for each bytecode to which native machine instructions ***will be generated***." ('205 patent at 13:15–17 (emphasis added).) Thus, while the table itself may be generated when the virtual machine is initialized, the native machine ***instructions*** of the snippet "will be generated" at a later time. As described above, the specification is clear that snippet generation—and the concurrent go_native generation—occurs ***during program execution***. (*See* '205 patent at 9:30–34; 7:49–51.) Thus, Figure 13 and its description in the patent does not support Oracle's proposed construction of the term "runtime," but rather is consistent with Google's proposed construction where runtime is "during execution of the virtual machine instructions."

4. Oracle's factual assertions are unsupported and inaccurate.

Oracle misses the point by proclaiming that the dispositive question is whether "the patent's claimed invention cover[s] the situation where the virtual machine is executing but is not necessarily executing virtual machine instructions." (Oracle Br. at 10.) This is a distinction without a difference: a virtual machine is executing when it executes virtual machine instructions. None of the evidence on which Oracle relies is to the contrary. For example, while Oracle cites to the '702 patent's description of the class initialization as an alleged example of when a virtual machine is executing but not executing virtual machine instructions (Oracle Br. at 12), the Court need look no further than another patent-in-suit, the '520 patent, to realize that this process involves executing virtual machine instructions.

Specifically, the '520 patent discloses that "the Java compiler generates a special method, <clinit>, to perform ***class initialization***, including initialization of static arrays" ('520 patent at 1:59–61 (emphasis added)), and then goes on to disclose the ***very bytecode*** (i.e., virtual machine instructions (*see* Oracle Br. at 11 (bytecode is synonymous with virtual machine instructions))) that the Java virtual machine executes to perform the class initialization. ('520 patent at 2:26–53 (Code Table #3).) Thus, Oracle's initialization example actually supports Google's proposed construction;[3] it proves that an executing virtual machine executes virtual machine *instructions*.

[3] Oracle's other examples (class loading, etc.) also involve execution of virtual machine instructions and support Google's construction. (*See, e.g.*, Fenton Decl. Ex. B at 2 ("The Java ClassLoader, furthermore, is written in the Java language itself.").)

DATED: January 11, 2012

KEKER & VAN NEST, LLP

By: /s/ Christa M. Anderson

ROBERT A. VAN NEST (SBN 84065)
rvannest@kvn.com
CHRISTA M. ANDERSON (SBN 184325)
canderson@kvn.com
KEKER & VAN NEST LLP
633 Battery Street
San Francisco, CA 94111-1809
Telephone: (415) 391-5400
Facsimile: (415) 397-7188

SCOTT T. WEINGAERTNER (*Pro Hac Vice*)
sweingaertner@kslaw.com
ROBERT F. PERRY
rperry@kslaw.com
BRUCE W. BABER *(Pro Hac Vice)*
bbaber@kslaw.com
KING & SPALDING LLP
1185 Avenue of the Americas
New York, NY 10036-4003
Telephone: (212) 556-2100
Facsimile: (212) 556-2222

DONALD F. ZIMMER, JR. (SBN 112279)
fzimmer@kslaw.com
CHERYL A. SABNIS (SBN 224323)
csabnis@kslaw.com
KING & SPALDING LLP
101 Second Street – Suite 2300
San Francisco, CA 94105
Telephone: (415) 318-1200
Facsimile: (415) 318-1300

IAN C. BALLON (SBN 141819)
ballon@gtlaw.com
HEATHER MEEKER (SBN 172148)
meekerh@gtlaw.com
GREENBERG TRAURIG, LLP
1900 University Avenue
East Palo Alto, CA 94303
Telephone: (650) 328-8500
Facsimile: (650) 328-8508

ATTORNEYS FOR DEFENDANT
GOOGLE INC.

I hereby certify that Christa M. Anderson joins in the e-filing of this document.

/s/ Cheryl A. Sabnis /s/
Cheryl A. Sabnis